Amendments to the Specification:

The Office Action requested that Applicants add a "Summary of the Invention" description to the application. However, Applicants would like to kindly point out that both the MPEP and 37 C.F.R. §1.73 do not require the presence of a "Summary of the Invention." They merely indicate where in the application the "Summary of the Invention" should be placed if Applicants choose to add one. 37 C.F.R. §1.73 only states that a "Summary of the Invention" should or may be included. It does not state "must" or "shall." Accordingly, Applicants have elected not to include a "Summary of the Invention" as this is within the discretion and right of the Applicants.

Please replace paragraph [0003] with the following amended paragraph:

[0003] Another characteristic of a processor architecture is whether instruction processing is pipelined. The processor fetches instructions from memory and sends them into one end of the pipeline in pipelined processing. The pipeline comprises ~~of~~ several stages, each of which perform some function necessary to process the instruction before the instruction proceeds to the next stage. Each stage moves the instruction closer to completion. A pipeline enables the processor to process more than one instruction at a time, thus increasing the instruction processing rate. Dependent instructions can cause a delay in the pipeline because processors typically do not schedule a dependent instruction until the instruction on which the dependent instruction depends has produced the correct result. But some pipelines process instructions speculatively. Speculative execution is where instructions are fetched and executed before pertinent data dependencies are resolved. During speculative execution, the processor predicts how dependencies will be resolved and executes instructions based on the predictions. The processor then verifies that the execution and predictions were correct before retiring the instruction and the results. Speculative execution can also involve predicting what instructions are needed depending on whether a branch is taken. For example, if two instructions are to be alternatively executed depending on the value of some quantity, then the pipeline has to predict what that value will be or which instruction will be executed. The processor then

predicts the next instruction to be executed and fetches the predicted instruction before the previous instruction is actually executed.

Please replace paragraph [0007] with the following amended paragraph:
[0007] **Figure 1** is a block diagram of a portion of a processor according to an embodiment of the present invention;

Please replace paragraph [0011] with the following amended paragraph:
[0011] A method and apparatus for a stopping replay tornadoes is disclosed. The embodiments described herein are described in the context of a microprocessor, but are not so limited. Although the following embodiments are described with reference to a processor, other embodiments are applicable to other integrated circuits or logic devices. The same techniques and teachings of the present invention can easily be applied to other types of circuits or semiconductor devices that can benefit from higher pipeline throughput and improved performance. The teachings of the present invention are applicable to any processor or machine that performs data speculation in executing instructions. However, the present invention is not limited to processors or machines that perform data speculation and can be applied to any processor and machine in which multiple levels of replay mechanism ~~is~~ are needed.

Please replace paragraph [0027] with the following amended paragraph:
[0027] Scheduler **30** is coupled to send instructions to execution units **40** and staging queues **50**. Scheduler **30** may re-order the instructions to execute the instructions out of program order to achieve efficiencies inherent with data speculation. In determining the order in which instructions will be sent to be executed, scheduler **30** uses the data dependencies of instructions and the latencies of instructions in determining the order of execution of instructions. Based on a combination of the data dependencies of various instructions and the anticipated execution time of instructions, the scheduler **30** determines the order of execution of instructions. The execution units **40** execute instructions and pass notification of things that have gone wrong, e.g. cache miss, to a

checker **60**. Staging queue **50** is used to send instruction information in parallel with execution unit **40** to the checker **60**. Staging queue **50** also supplies 37 the instruction to the replay multiplexor **35**, which can put the instruction back into execution if so ordered by the checker **60**.

Please replace paragraph [0028] with the following amended paragraph:

[0028] The checker **60** identifies primary replay cause instructions from notification from the execution units **40**. The checker then determines secondary replaying instructions by tracking the propagation of erroneous results from instruction to instruction. The checker **60** will pass correctly executed instructions to retire unit **70**. The checker **60** can direct the replay multiplexor **35** to place an instruction that has not produced correct results back into execution immediately. When this is done, the scheduler **30** is prevented from dispatching an instruction that would collide with a replaying instruction in the replay multiplexor **35**. The checker **60** also has the option of simply letting an instruction that has not produced correct results disappear; that is, the instruction does not go to the retire unit **70** or back into execution via the multiplexor **35**. In any case, the checker **60** needs to report 22 to the replay queue **20** that the instruction is replay safe or not. The instruction is replay safe if the checker **60** has determined that it produced the correct result and has passed it to the retire unit **70**. Retirement unit **70** is coupled to checker **60** and retires the instructions in program order and applies the results to the architectural state if no errors exist. Upon retirement, certain resources are to be reclaimed from the instruction. The retire unit **70** informs the allocator/renamer **10** of the retirement so that they may reclaim appropriate resources. Depending on the embodiment, it may be necessary for the replay queue **20** to also be informed of the retirement of an instruction.

Please replace paragraph [0032] with the following amended paragraph:

[0032] When replay queue **20** makes all instructions that are not replay safe eligible to schedule, the net effect to instruction scheduling is that all memory that some of these instructions have been in the scheduler before or have executed before ~~is~~ are gone. All instructions that are not replay safe are assumed by the scheduler to have never produced

correct results before. All must now be sent to execution and the scheduler **30** will speculate only on the results of this new execution.

Please replace paragraph [0035] with the following amended paragraph:

[0035] When the checker **60** has determined that an instruction has not executed correctly, there are two distinct ways in which the instruction can be executed again (replayed) to try again to get the instruction correctly executed. The first method is called a "fast replay". In the first method all information necessary to execute the instruction again is available from the staging queue **50**. The checker **60** can switch the multiplexor **35** to reinsert **36** this information immediately back to execution. The scheduler **30** has to be prevented from dispatching a new instruction to execution that would collide with the instruction being replayed. A second method is called a "replay from the replay queue". In the second method, no action is taken immediately. Because the instruction was not declared "replay safe" by the checker **60**, the instruction is still "not replay safe" in the replay queue **20**. The instruction simply disappears from the execution pipe since the checker **60** has not passed it to retirement and has not reinserted it for immediate re-execution as in the first method. At some later time, all instructions in the replay queue **20** that are not "replay safe" will be made "eligible to schedule". This will cause this instruction, when its turn comes in program order, to be sent again to the scheduler **30** and from there to execution. The first method has the advantage that an instruction can be executed again immediately upon discovering that its last execution did not produce a correct result. Usually it can be highly beneficial to performance to re-execute an instruction that was not executed correctly very quickly, specifically at the time that correct data could be available as a result of access to a higher level cache. This is because a cache miss is the most common event to cause incorrect results. The first method, however, provides little flexibility. The second method takes much longer to return an instruction that was not executed correctly to try executing it again. The advantage is that the second method provides a high level of control on how that is done. Replaying from the replay queue, the second method, as has been mentioned before, will

not produce a tornado and will, in fact, stop any tornadoes that may have otherwise started.

Please replace paragraph [0041] with the following amended paragraph:

[0041] **Figure 2** is a block diagram of a processor **2** that includes an embodiment of a tornado detection and replay mechanism **20** in accordance the present invention. The front end **4** of the processing phase is coupled to the scheduler **30** via allocator/renamer **10** and a rescheduler replay queue **20**. Instructions are dispatched speculatively from the scheduler **30**. Thus the scheduler **30** can dispatch an instruction without first determining whether data needed by the instruction is valid or available. Scheduler **30** dispatches instructions from the front end when the input data needed to execute the instructions is projected to be ready when the instruction would reach execution. The scheduler includes a counter 32.

Please replace paragraph [0057] with the following amended paragraph:

[0057] **Figure 4** is a flow chart showing one embodiment of a method of processing instructions in accordance with the present invention. At block **402**, the processor pipeline fetches instructions from memory. The instructions are decoded at block **404**. The processor then allocates the necessary processor resources at block **406** for each decoded instruction. The instructions are placed in a scheduler queue at block **408** for scheduling. Each instruction that is ready for execution is scheduled or dispatched for speculative execution at block **410**. At block **412**, each instruction is speculatively executed. After the instructions are executed, the instructions and results are checked for errors and dependency issues at block **414**. At block **416**, a determination is made as to whether an error occurred. If no error exists and the instruction executed correctly, the instruction is retired at block **418**. The results of that instruction are committed to the architectural state at block **420**. But if an error exists, then the instruction is transmitted to the replay multiplexor at block **422**. The instructions that encountered execution problems are replayed at block **424**. A tornado detection mechanism monitors the replay for the presence of a tornado. If a tornado is detected, the replay is stopped and the

tornado broken at block 426. A tornado safe replay is initiated for the stopped instructions. A replay from the replay queue is an example of a tornado safe replay. The tornado detection mechanism records and tracks the instructions that cause replay tornadoes 428. A table of these troublesome instructions can be used by a tornado prediction mechanism to avoid future tornadoes.